A Methodology to Transform an OS-based Application to a Bare Machine Application

Uzo Okafor, Ramesh K. Karne, Alexander L. Wijesinha, and Patrick Appiah-Kubi
Department of Computer and Information Sciences
Towson University
Towson, MD 21252, USA
uokafo1@students.towson.edu, {rkarne,awijesinha,appiahkubi}@towson.edu

***Abstract*—This paper describes a novel approach to transform application programs that run with the support of an operating system or kernel to bare machine applications that run with no intermediary software of any kind in the machine. The general transformation methodology is based on a simple model that views application software as code with system header files and calls. These files and underlying system calls are removed in order to transform the application without understanding details of the application or its internal behavior. A Windows SQLite database engine application is chosen to illustrate the transformation process, and the transformed application is run on a bare PC to demonstrate the feasibility of this approach. The Microsoft Windows Visual Studio environment (IDE) is used to facilitate the transformation process. Sample database queries are run on a bare PC, and also in Visual Studio, and the results are validated by comparison. Currently, the application code is transformed manually; however, the experiences and skills acquired could be leveraged to develop an automated tool in the future. This methodology and transformation model serves as a basis for converting a variety of applications and other software to run on bare machines by achieving automatic ubiquity without a need for a virtual machine. When the general transformation methodology is fully tested, made robust, and automated, existing computer software can be transformed with little effort to make them independent of any operating environment.**

***Index Terms*—SQLite, bare PC, operating system, transformation models, bare machine computing.**

I. INTRODUCTION

Application programs written in a programming language are translated to a machine code by a compiler based on the underlying machine architecture and operating system (OS) environment. Each program also needs I/O or system calls/libraries to access hardware resources. An operating system acts as a form of intermediary software to provide hardware abstractions to application programs. Thus, application programs are not truly independent of their execution environment, and are susceptible to rapid changes in operating systems, distributions, kernel versions, and computing platforms. If an application program can be made bare (i.e., totally independent of its execution and operating environment), essentially the same code can be run on a variety of devices, including pervasive devices thus making it ubiquitous without the need for a virtual machine. It will also then be possible to write programs that are more easily adapted to advances in hardware technology. The result is a Bare Machine Computing (BMC) paradigm that is application-centric rather than OS and platform-centric. While Java and other virtual machines provide portability and ubiquity, the applications running in these environments are not bare.

Alternatively, an application program may be viewed as being intertwined with the OS and the underlying machine's CPU architecture. The system calls inserted by the compiler are provided by the OS (in addition to other hardware abstractions), and the high-level language translation to machine code is dependent on the underlying machine architecture. A preliminary effort to transform a SQLite application is outlined in [15]. It provides a high-level transformation methodology and classified system calls. However, it does not give complete details of the transformation process, and it does not provide a methodology that could be applied to transform other applications to run on bare machines. This paper describes a general transformation methodology for eliminating hardware abstractions (OS/kernel, system libraries, or other forms of intermediary system software) via hardware interfaces directly accessible from application programs [9]. The transformation process is illustrated in detail for the case of SQLite, and test results after transformation are presented.

For the purposes of transformation, application software is modeled simply as code that undergoes resolution of system calls/libraries during the compilation/link process so that appropriate bare machine interfaces to the hardware can be included with the application itself. Currently, this process is done manually as illustrated in this paper. Eventually, an automated tool can be developed that can transform a variety of applications to run on bare machines. Several bare PC or BMC applications have been built and shown to outperform their OS-based counterparts. These include a Web server [7], VoIP softphone [12], split protocol server [17], SIP server [1], and Webmail server [3]. A generic BMC concept and methodology for developing bare applications is described in [13]. In addition, the design and implementation of a bare PC USB driver and file system are described in [10]. However, each of these was built directly, and not by transforming existing OS-based software.

Compilers, interpreters, programming language parsers, and other tools used for porting applications (to run on different platforms) use well-known translation methodologies. Examples of software and tools that perform code transformation and translation, or enable applications to run directly on the hardware (with the support of some form of an OS such as a kernel, or as an embedded system) include [2] [4] [8] [18]. In contrast, the BMC paradigm uses an application

object (AO) [11], which bundles a related suite of applications coded as a single monolithic executable so that they can work together without any supporting OS or kernel. The BMC paradigm is in essence similar to approaches such as Exokernel [5], IO-Lite [16], and Palacios and Kitten [14], whose primary goal is to eliminate overhead and complexity due to OS abstractions. However, the BMC approach is at the extreme end of eliminating OS or kernels. Also, the main difference in a bare application is that the necessary code to interface with the hardware is included within the application itself. This gives full control over the hardware to the application software developer, whose code is now responsible for allocating and managing resources (CPU/memory), and scheduling (processes/tasks). A cost/benefit analysis of using lean or "barebones" versus feature-rich systems is given in [19].

II. SQLITE APPLICATION

SQLITE is a popular standalone single user database engine that runs on Windows or Linux. It was selected to demonstrate the transformation methodology since it is a large C programming application and consists of complex structures and a combination of styles. The Windows version is in the form of an amalgamated package that consists of two files; shell.c and sqlite3.c. The sizes of shell.c and sqlite3.c are 86, 016; 4,323,826 bytes respectively. The total number of lines of code in both source files is 129,003 with 55,691 lines of commented code and 40,297 executable statements. The code is complex: there are over 5000 cases in a particular switch statement, hundreds of macros, and numerous user-defined OS-related functions; also, dozens of pre-processor statements are part of the code. The application supports standard database functions such as create tables, insert data and query a database. The output is displayed in a Microsoft Window (there is no graphics interface).

III. TRANSFORMATION PROCESS

The SQLITE code needs to be transformed so that it can be run on a bare PC. There are many challenges when transforming large OS-based applications with complex code to run on a bare PC (testing, validation and debugging can pose problems especially since there is no environment such as an IDE to support bare application development, and only a few primitive tools that can run on a bare system). Some issues are partially resolved by using the Visual Studio (VS) environment for testing, validating and debugging the code during the transformation process. The VS environment does not allow the complete bare machine code to run, as it requires resources from its Windows environment through system calls. It is difficult to eliminate all system calls when using the VS/Windows platform. For example, memory allocation (malloc()) uses virtual memory and it is obtained from heap space. In a bare PC, this is physical memory; it is allocated and controlled via the AO code by the bare software developer (the file system is also managed by the application if it is required). There are hundreds of header files included in a Windows program, even if the application program does not require all of them. The header file "Windows.h" is an example for this.

The general transformation process model is shown in Fig.1. VS 10 (C/C++ compiler) is used as the development platform for the bare PC application, with batch files to compile bare PC programs. The bare PC hardware API to support the application is also built during the transformation process (some interfaces may be reused from other bare applications). The main objective behind the transformation approach is to eliminate OS dependencies without understanding code details relevant to application logic, and to minimize changes to the original code.

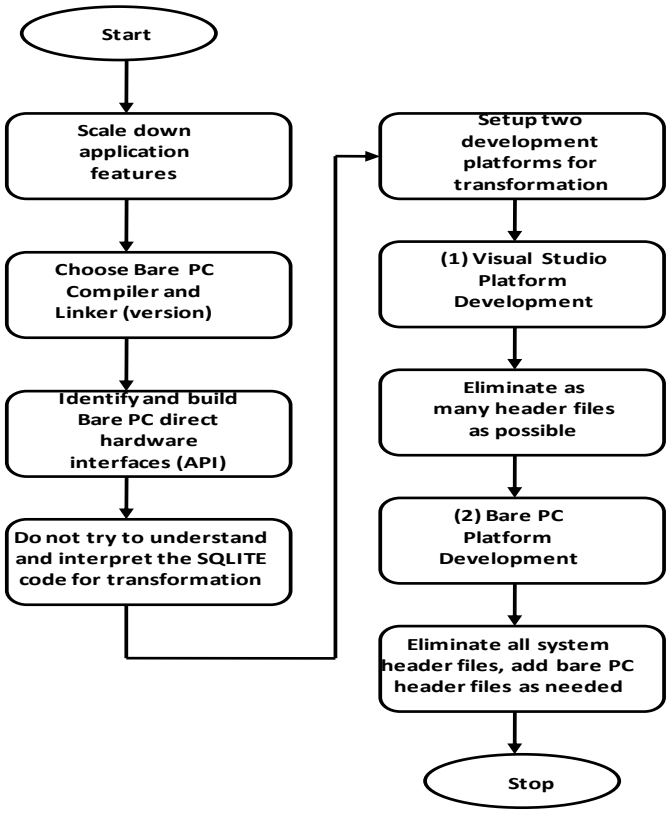

Fig. 1. Transformation Methodology

Two development environments were set up: one for the VS application and the other for the bare PC application. In the VS application, as many header files as possible are eliminated; this code is then used to transform to a bare PC. Then the remaining system header files are removed by adding the bare PC interfaces and fixing any bare PC-related issues until the application runs successfully on a bare PC. This means the bare PC application now runs and has the same results as the OS application. More details regarding the transformation process are given below.

A. Scaling Down Features

By scaling down some functionality in SQLITE, the transformation process is simplified. For example, an "**in-memory**" database was used to eliminate file-related code in the transformation. Floating point and shared cache options were also turned off. Complex concurrency and locking mechanisms were not used since these are avoided in the BMC paradigm (it is possible for multiple tasks to run concurrently in

bare PC applications such as [3] [7] [17]). The scaled-down options apply to both the VS and the bare PC SQLite applications during transformation.

B. Visual Studio Application

For the VS application, baseline code was downloaded from the SQLITE Web site and scaled down as noted above. A test case suite was developed to test query results that includes standard commands such as create table, insert (multiple) rows, and select table. This suite was used to test correctness of database operations after each step of the transformation in VS. Fig. 2 shows the steps in this transformation process, whose goal is to remove as many OS dependencies as possible.

VS offer numerous compile and link options that help to transform an application so that it runs on a bare PC. For example, the **NODEFAULTLIB** option is used to identify system calls that are in the application. All these system calls must be resolved in the bare PC application to make it run without OS support. There were 85 such system calls in SQLITE. There were additional dependencies due to Link options, which were handled by including a bare PC user library to be used during compilation i.e., bare PC direct hardware interfaces were put in a library (**rkkvs.lib**) to replace the original system libraries. The Assemble Machine Code and Source Listing option, **/FAcs**, is used to generate the assembly listing and asm files (this is very useful to understand the role of system calls in the code).

The process to transform the VS application is similar to that of developing an ordinary C/C++ application (except that the main focus of transformation is to eliminate all system calls/libraries). One header file at a time in the VS application is removed by commenting it in the source code. When the program is compiled and linked, it shows the missing system calls in the application. These calls are replaced with bare PC direct hardware interfaces and recompiled. Only one system call at a time is resolved since the bare PC interfaces are not yet fully tested. When the bare hardware API becomes more robust, it will be possible to resolve multiple (or all) system calls together to speed up the transformation process. The VS application uses a large number of libraries and DLLs. It is necessary to guarantee that the system calls handled by the bare hardware API are used by the linker.

These system calls or interfaces used come in three forms: **(1)** the name of the call has single underscore and it is explicit in the code (e.g strcmp(), _strcmp()): in this case, all strcmp() methods in the source code must be replaced by AOAstrcmp() to guarantee the usage of the bare API; **(2)** the name of the system call has a double underscore and it is not explicit in the code (e.g. __allmul): in this case, this call must be added to the bare PC library and it must be removed from the system libraries; **(3)** the name of the call sometimes is simply a constant such as __fltused: in this case, that constant and its value are provided if needed. During the transformation, system header files are removed and bare PC hardware API are added until no more header files can be removed. Some header files invoke other header files and invocations may be indirectly recursive. For example, it is not possible to remove the "windows.h" file during VS compilation.

In general, every system call needs a header file, but every header file may not have a system call. There can be different types of header files: e.g., user header files, constant header files, and structure header files. The 85 system calls in the SQLITE application, as shown in Fig. 3, were classified into the following types: 8 Arithmetic Assembly, 2 Disk Management, 2 Standard I/O, 1 Error, 27 File Function, 2 Floating Point, 1 Object Handle, 3 Library, 10 Memory, 3 Process, 1 Stack, 4 String, 2 System, 10 Timer, 7 Type, and 2 Unicode and Character Set calls. According to [6] and [20] respectively, Linux and Windows systems have three to four hundred system calls.

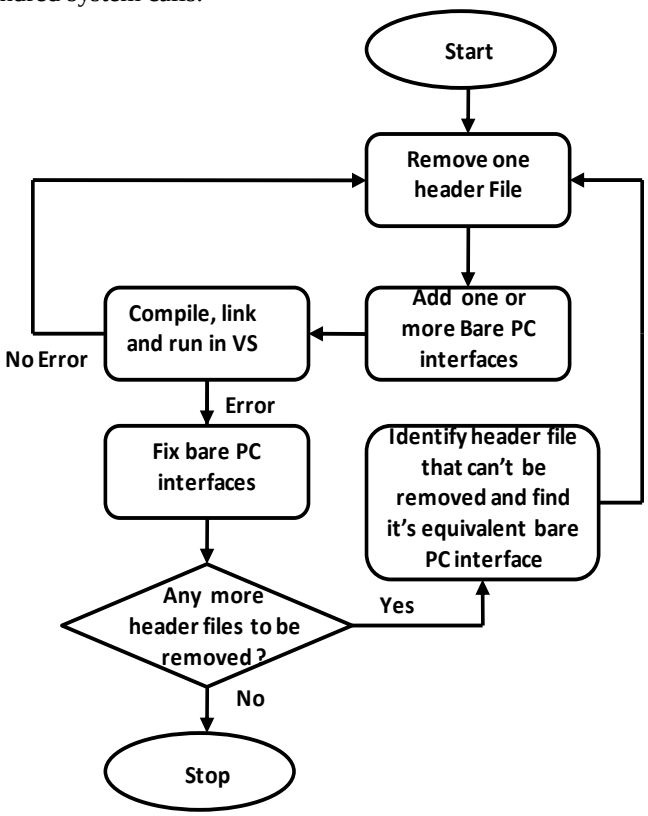

Fig. 2. Visual Studio (VS) application

The header files that were not removable were: windows.h, stdarg.h, stdio.h, stdlib.h, and assert.h. Since we scaled down the application, we were able to provide all bare PC hardware interfaces except for malloc(). In VS, malloc() call gets memory from memory management and it is a virtual memory with paging. In a bare PC, all memory is real and the AO programmer manages it at program time. There were a large number of malloc() calls used in the code. We used the bare PC memory object to replace all the calls except for a single system malloc() to obtain memory as required by the application. The total memory obtained by this call was used by the bare PC malloc object to allocate, reallocate, and free memory. The pseudo bare PC code for SQLITE was then tested to verify that its results were the same as for the original VS code. Next, the

transformed VS code was transferred to the bare PC to determine other modifications that were needed.

C. Bare PC Application

The transformation process on a bare PC is shown in Fig. 4. During the pseudo transformation in VS, system calls such as malloc() transferred from the VS system were eliminated and replaced with calls in the bare PC API. There were also other header files that could not be removed in the VS application.

Again, one header file at a time was removed from the list (windows.h, stdarg.h, stdio.h, stdlib.h, and assert.h) and replaced with appropriate bare PC interfaces until all header files were removed. The transformed SQlite application was compiled, run and tested after each modification of the code to remove the remaining header files. In some cases, header file removal required a new header file in the bare PC application that defines some constants, variables, structures, and data. In the SQLite transformation, we defined header files such as sqlite.h, sqlite21.h, and stdarg.h to cover some required definitions. In this case, once header files were removed and the VS system malloc() (just one call) was handled, the application successfully ran in a bare PC. To verify success, we tested the bare PC application with a variety of create, insert and select statements and validated their functionality. We also checked that the results matched with the original VS application model for every instance of testing.

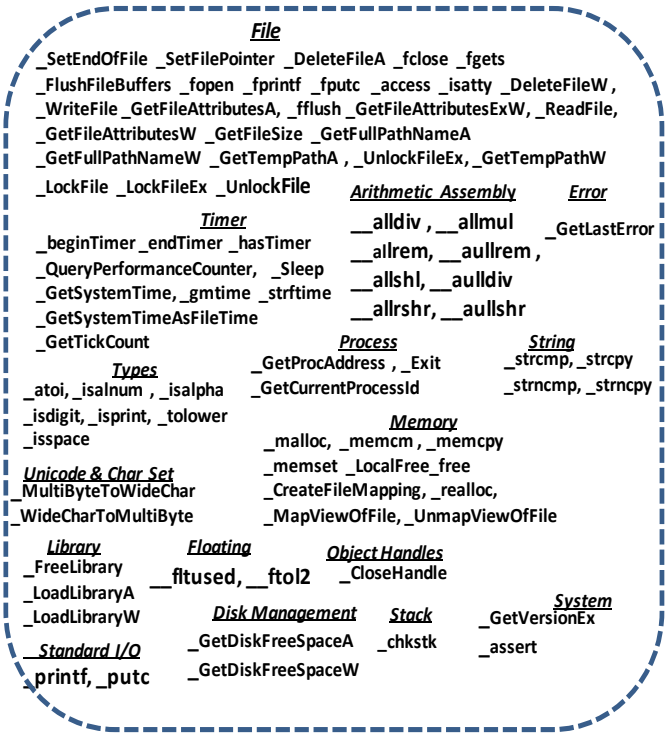

Fig. 3. System calls

In general, once all header files are removed and replaced with appropriate interfaces, it should be possible to successfully run the application in a bare PC. If not, then there must be some issues with the bare PC interfaces or the transformation itself. Usually, these arise from system problems related to the bare PC executable module and its memory layout. They may also be related to the bare PC loader and the BIOS, or the processing of system interrupts. While it cannot be proven that this transformation methodology will work in a general case, its success in transforming the SQLite application indicates that this is a correct approach to modeling application software to be transformed i.e., it is not necessary to understand any code details in the original application during the transformation process.

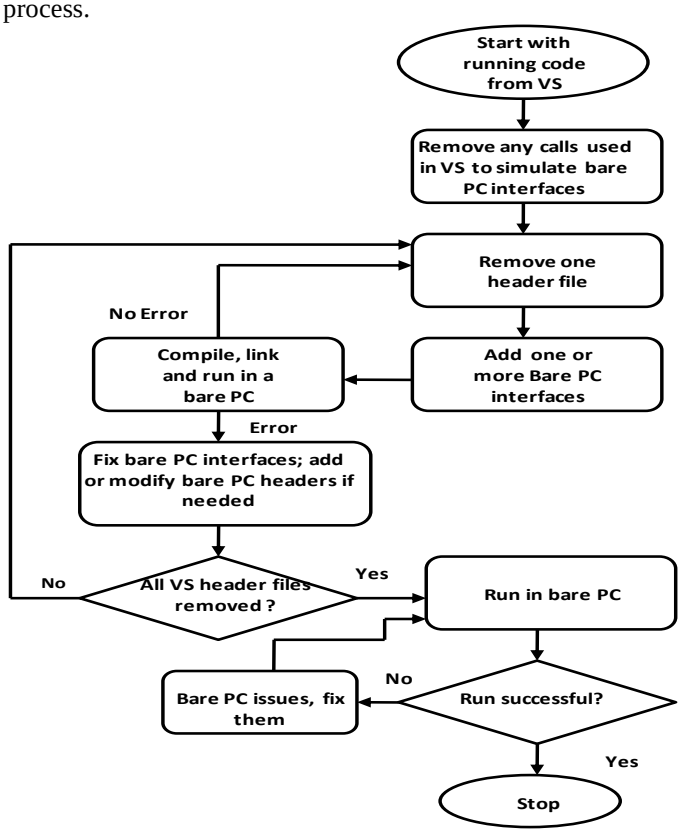

Fig. 4. Bare PC application

The **/bin** files from Visual Studio 10 Express Version are used for compilation and linking of the transformed bare PC application. The actual make file used to compile, link and generate a bootable USB for running the bare PC SQLite application is shown in Fig. 5. Most of the statements in the figure are self-explanatory, except for the last statement. The **rwhd.exe** module installs a bare PC boot record after all other files are copied onto the USB. The USB is also formatted before copying the files. The **prcycle.exe** file is the startup menu for bare PC applications, and the **shell.exe** file is the actual bare PC application (main component of the AO) for SQLite. The **data.txt** file is a data file that can be used in the application for receiving additional parameters from the user. One can also store the persistent database files on the USB after they are used. The persistent database files are not shown here as we only demonstrated the SQLite application with an "**in-memory**" database. A full-scale version of the SQLite

transformation would include persistent storage containing the database schema and data.

```
..\bin\ml /c /Cx /Fl  asmfiles.asm asmfilesb.asm chkstk.asm

..\bin\cl /c /FA /Zl /Os /Gy /GS- shell.c sqlite3.c cfiles.c aaa.c
aaab.c memobj.c

...\bin\link /MAP /BASE:0x00100000 /NODEFAULTLIB
/OPT:NOREF /MERGE:.rdata=.data /OPT:NOICF
/STACK:32000000 /LIBPATH:"rkkvs.lib" /ENTRY:main shell.obj
sqlite3.obj cfiles.obj aaa.obj aaab.obj asmfiles.obj
asmfilesb.obj memobj.obj chkstk.obj

format e: /FS:FAT32 /q

copy prcycle.exe e:
copy shell.exe e:
copy data.txt e:

rwhd -m 5 newbootf32.bin
```

Fig. 5. Batch files

```
        SQLITE Trasformation, Towson University                  0000000B
01 1        2        3        4        5        6        7        8
02 create table t100(c1 int, c2 int, c3 char(20), c4 char(10), c5 int);
03 insert into t100(1111, 2222, 'Karne', 'Okafor', 1000);
04 insert into t100 values(1111, 2222, 'Karne', 'Okafor', 1000);
05 insert into t100 values(2222, 3333, 'Wang', 'Peter', 2000);
06 insert into t100 values(3333, 4444, 'Mike', 'Wijesin', 3000);
07 insert into t100 values(4444, 5555, 'Borg', 'Taylor', 4000);
08 create table t200(c200 int, c201 char(30));
09 insert into  t200(8888, 'Table2');
10 insert into t200 values(8888, 'Table2');
11 insert into t100 values(5555, 6666, 'Eyer', 'Matyas', 5000);
12 select * from t100;
13 sqlite> _

15 c1       c2       c3       c4       c5
16 1111     2222     Karne    Okafor   1000
17 2222     3333     Wang     Peter    2000
18 3333     4444     Mike     Wijesin  3000
19 4444     5555     Borg     Taylor   4000
20 5555     6666     Eyer     Matyas   5000
21
22
23       00000000  1
24 00000000        C2  D2  E2         00000000  1400D12B  00000000         T
```

Fig 6. Bare PC output display (simple queries)

IV. RESULTS AND DISCUSSION

Fig. 6 shows simple queries consisting of Create, Insert and Select statements. It demonstrates the creation of a table (t100) with 5 columns and different data types in line 02. Line 03 shows an insert statement, with an error as it does not have the **"values"** keyword in the statement. This demonstrates the ability of the parser to detect a syntax error. Lines 04-07 show inserting more values into the table successfully. Line 08 shows the creation of a different table (t200). Line 09 shows an attempt to insert data into this table that has a syntax error, and Line 10 shows a correct insert. Line 11 shows more inserts into the t100 table. Line 12 shows the select statement for the t100 table. Lines 15-20 correctly print the query results for the t100 table. Lines 23-24 show some output for debugging and testing the bare code. It can be seen that the bare PC screen is divided into 8 columns and 24 rows to display the SQLite output. This is currently a text-based window similar to **"stdout"**.

```
        SQLITE Trasformation, Towson University                  00000009
01 1        2        3        4        5        6        7        8
02 create table employee(name char(15), ssn int, addr char(20), dno int);
03 create table dept(dname char(10), mssn int, dnum int);
04 insert into employee values('Smith', 1111, 'MD', 1);
05 insert into employee values ('Wang', 2222, 'VA', 2);
06 insert into employee values('Okafor', 3333, 'NJ', 2);
07 insert into employee values('Lee', 4444, 'NY',2);
08 insert into dept values('Research', 1111, 1);
09 insert into dept values('Admin', 2222, 2);
10 select name, addr from employee, dept where dname='Admin' and dnum=dno;
11
12
13 sqlite> _

15 name       addr
16 Lee        NY
17 Okafor     NJ
18 Wang       VA
19
20
21
22        dept       employee
23 0      00000000   1
24 00000000        C2  D2  E2         00000000  1400D12F  00000000         T
```

Fig 7. Bare PC output display (complex queries)

Fig. 7 shows a complex query consisting of the creation of two tables and insertion of some data. The complex query on Line 10 shows a join and the printing of some attributes. The results of this query are correctly shown on Lines 15–18. The **.tables** meta-command in SQLITE is also tested and its results are shown on Line 22 (two table names: dept and employee). The sample screen output shown in Figs. 6 and 7 demonstrate the correct functionality of the transformed code that runs on a bare PC. We also tested more queries and meta-commands to validate the correct functionality of the transformed code.

These results serve to verify that this transformation approach using the VS IDE to test, validate and debug bare PC code is a viable approach. Since the model and methodology are very general, it is expected that they can be used to transform other complex applications. The scaled-down approach used in this paper needs to be extended to transform the full SQLite application with a file system and other features. The bare PC hardware API also needs to be enhanced

to deal with other components of the standalone database engine and multi-threaded applications. The transformation process and its feasibility demonstrates that it is possible to make existing applications to run on bare machines thus achieving a different form of ubiquity without using virtual machines. This observation infers a great potential for existing applications to make them independent of OS or environments when an automated tool is made available in the future.

V. CONCLUSION

This paper presented a novel approach to transforming a conventional application so that it runs on a bare PC with no OS or kernel. The proposed transformation methodology is based on a simple software model. The key idea is to remove header files and replace them with equivalent bare PC interfaces and add only required header files for bare PC. The methodology was used to transform a Windows SQLite application to a bare PC application with an "**in-memory**" database.

First, the VS platform and IDE was used to pseudo transform code from Windows to a bare PC application. Thereafter, minimal modifications to handle non-removable system files produced the complete bare PC application code. It was not necessary to modify the SQLite application code, or understand the underlying application logic or the internal details of database structures. The transformed bare PC application was tested and validated for correct operation by running sample queries in a VS (Windows) environment, and on a bare PC, and comparing the results. The methodology may be used to transform other complex applications to bare PC applications, making the code independent of any OS or environments. The currently manual transformation process could be modified in the future to build a tool for automatically transforming C/C++ or other programming applications. The methodology will also serve as a basis to transform applications that can run on a variety of systems and devices. The initial successful transformation of a complex application such as SQLITE indicates that future research into developing applications that run without OS or kernel support for performance, security, or other reasons may benefit from the new transformation methodology.

ACKNOWLEDGMENT

The authors are grateful to the late Dr. Frank Anger of NSF for his encouragement and support of early bare machine computing research, supported by NSF SGER grant CCR-0120155.

REFERENCES

[1] A. Alexander, R. Yasinovskyy, A. L. Wijesinha, and R. K. Karne, "SIP Server Implementation and Performance on a Bare PC," International Journal in Advances on Telecommunications, vol. 4, no. 1 and 2, 2011.

[2] W. Ahn, S. Qi, M. Nicolaides, and J. Torrellas, "BulkCompiler: High-Performance Sequential Consistency through Cooperative Compiler and Hardware Support," IEEE/ACM International Symposium on Micro Architecture, 2009.

[3] P. Appiah-Kubi, R. K. Karne, and A.L. Wijesinha,"The Design and Performance of a Bare PC Webmail Server," The 12th IEEE International Conference on High Performance Computing and Communications (HPCC) AHPCN, 521-526, 2010.

[4] R. Cáceres, C. Carter, C. Narayanaswami, and M. Raghunath, "Reincarnating PCs with Portable SoulPads", IBM T.J. Watson Research Center, New York.

[5] D. Engler, "The Exokernel Operating System Architecture," Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Ph.D. Dissertation, 1998.

[6] FreeBSD/Linux Kernel Cross Reference, http://fxr.watson.org/fxr/source/kern/syscalls.c, retrieved Feb 16, 2012.

[7] L. He, R. K. Karne, and A. L. Wijesinha, "Design and Performance of a bare PC Web Server," International Journal of Computer and Applications, vol. 15, 100-112, Acta Press, 2008.

[8] Y. Hwang, T Lin., R. Chang, "DisIRer: Converting a Retargetable Compiler into a Multiplatform Binary Translator. In ACM Transactions on Architecture and Code Optimization," vol. 7, issue 4.

[9] R. Karne, R., K. Jaganathan and T. Ahmed, "How to run C++ Applications on a bare PC," 6th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel Distributed Computing (SNPD), 50-55, 2005.

[10] R.K. Karne, S. Liang, A.L. Wijesinha and P. Appiah-Kubi, "Bare P**C** Mass Storage USB Driver," International Journal of Computer and Applications, March 2013.

[11] R. K. Karne. "Application-oriented Object Architecture: A Revolutionary Approach," 6th International Conference, HPC Asia, 2002.

[12] G. H. Khaksari, A. L., Wijesinha, R. K., Karne, L., He and S. Girumala, "A Peer-to-Peer Bare PC VoIP Application," IEEE Consumer and Communications and Networking Conference (CCNC), 803-807, 2007.

[13] G. H. Khaksari, A. L. Wijesinha, and R. K. Karne. "A Bare Machine Development Methodology," International Journal of Computer Applications (IJCA), vol. 19, no.1, 10-25, 2012.

[14] J. Lange et al., "Palacios and Kitten: New High Performance Operating Systems for Scalable Virtualized and Native Supercomputing," Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium, 2010.

[15] U. Okafor, R. K. Karne, A. L. Wijesinha, and B. S. Rawal, "Transforming SQLITE to run on a bare PC," ICSoft, 2012.

[16] V. S. Pai, P. Druschel and Zwaenepoel, "IO-Lite: A Unified I/O Buffering and Caching System," ACM Transactions on Computer Systems, vol.18 (1), 37-66, 2000.

[17] B. Rawal, R. K.Karne, A. L. Wijesinha, "Mini Web Server Clusters for HTTP Request Splitting," IEEE International Conference on High Performance Computing and Communications, 94-100, 2011.

[18] M. Schoeberl, S. Korsholm, T. Kalibera, and A. P. Ravn, "A Hardware Abstraction Layer in Java," ACM Transactions on Embedded Computing Systems, vol.10, no. 4, Article 42, 2011.

[19] S. Soumya, R. Guerin, and K. Hosanagar, "Functionality-rich vs Minimalist Platforms: A Two-sided Market Analysis," ACM Computer Communication Review, vol. 41, no. 5, 36-43, 2011.

[20] Windows System Call Table, Googlecode.com, retrieved Feb 16, 2012. http://miscellaneuoz.google.com/svn/winsyscall.